Object-Orientation Overview

B. W. Bush
Energy and Environmental Analysis Group
Los Alamos National Laboratory
8 October 1996

Outline

- *definitions*
- *key features*
- *development process*
- *analysis*
- *design*
- *programming*
- *bibliography*

A copy of these viewgraphs and additional notes is available at “http://bwb.lanl.gov/bwb.htm”.

Definitions

- *"Something is* **object-oriented** *if it can be extended by composition of existing parts or by refinement of behaviors. Changes in the original parts propagate, so that compositions and refinements that reuse these parts change appropriately." [Goldberg]*
- *An* **object** *has state, behavior, and identity. "An* **object** *is characterized by a number of operations and a state which remembers the effect of these operations." [Jacobson]*
- *"A* **class** *represents a template for several objects and describes how these objects are structured internally. Objects of the same class have the same definition both for their operations and for their information structures." [Jacobson]*
- *"An* **instance** *is an object created from a class. The class describes the (behavior and information) structure of the instance, while the current state of the instance is defined by the operations performed on the instance." [Jacobson]*

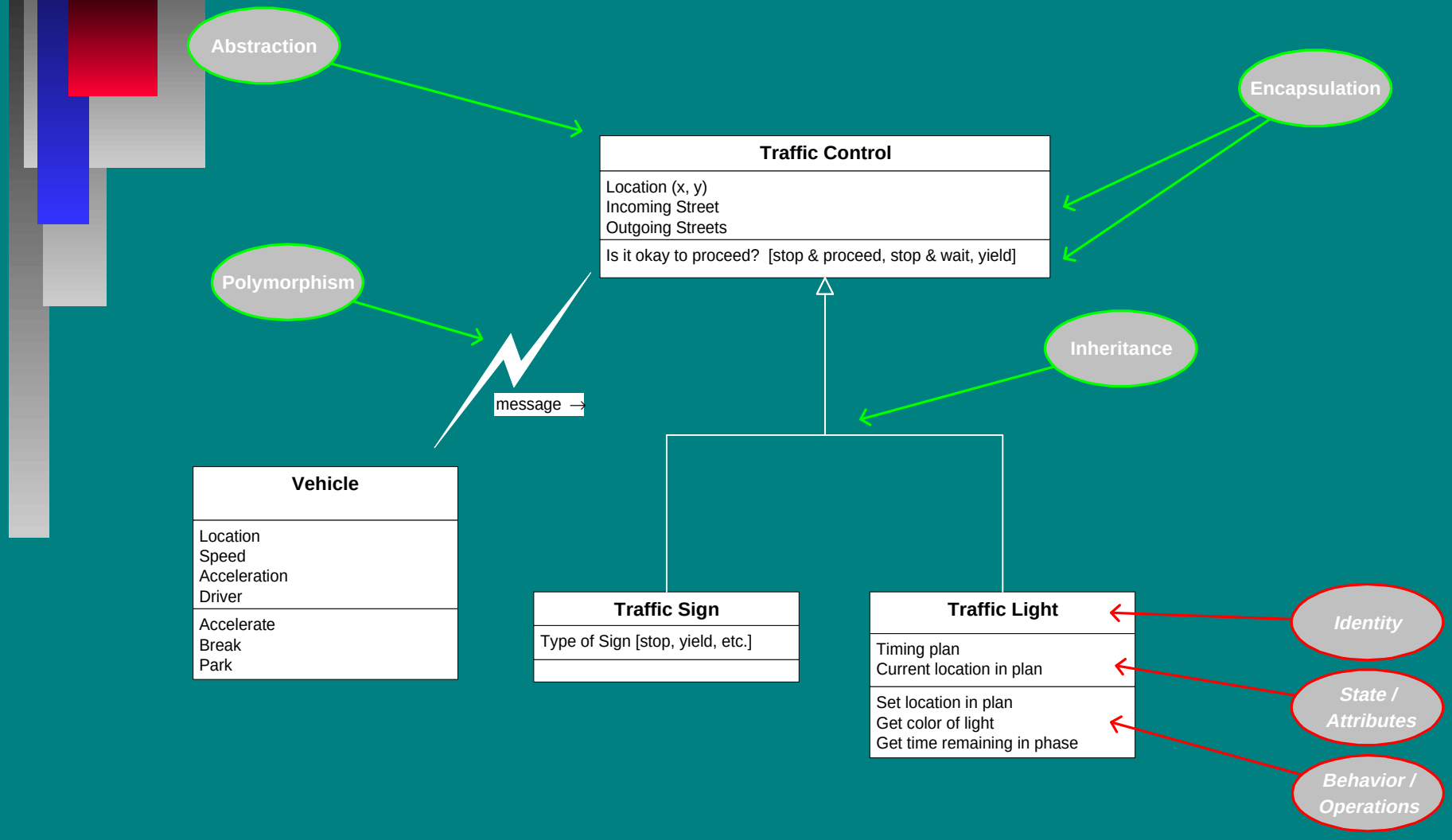
Key Features

- *abstraction*
  - *"An* **abstraction** *denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer." [Booch]*
  - *allows building models which map to the real world*
- *encapsulation*
  - *"***Encapsulation** *is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation." [Booch]*
  - *hides implementation details*

Key Features (continued)

- *inheritance*
  - *"If class B* **inherits** *class A, then both the operations and the information structure described in class A will become part of class B." [Jacobson]*
  - *enables and organizes code reuse*
- *polymorphism*
  - *"***Polymorphism** *means that the sender of a stimulus does not need to know the receiving instance's class. The receiving instance can belong to an arbitrary class." [Jacobson]*
  - *reduces software maintenance and increases extensibility.*

Example


Abstraction
Encapsulation
Traffic Control
Location (x, y)
Incoming Street
Outgoing Streets
Is it okay to proceed? [stop & proceed, stop & wait, yield]
Polymorphism
message
Inheritance
Vehicle
Location
Speed
Acceleration
Driver
Accelerate
Break
Park
Traffic Sign
Type of Sign [stop, yield, etc.]
Traffic Light
Timing plan
Current location in plan
Set location in plan
Get color of light
Get time remaining in phase
Identity
State / Attributes
Behavior / Operations

Usefulness of Object-Orientation

- *issues addressed*
  - *scheduling: meeting delivery dates*
  - *complexity: modeling complex applications*
  - *size: managing interdependencies in large systems*
  - *compatibility: making different chunks of code inter-operate*
- *benefits*
  - *reuse of code*
  - *reduced code size*
  - *increased productivity*
  - *lower defect rate*

Successful Projects

- *"The five habits of a successful object-oriented project include:*
  - *"A ruthless focus on the development of a system that provides a well-understood collection of essential minimal characteristics.*
  - *"The existence of a culture that is centered on results, encourages communication, and yet is not afraid to fail.*
  - *"The effective use of object-oriented modeling.*
  - *"The existence of a strong architectural vision.*
  - *"The application of a well-managed iterative and incremental development life cycle." [Booch]*

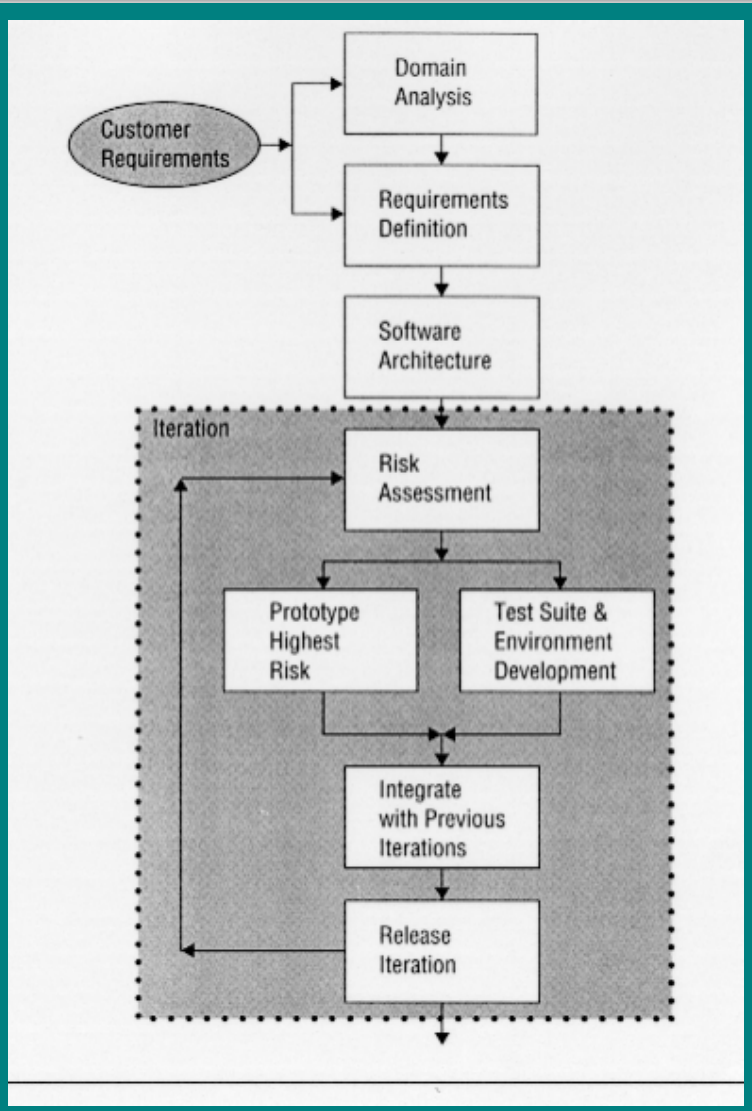
Successful Projects (continued)

- *"Why do certain object-oriented projects succeed? Most often, it is because:*
  - *"An object-oriented model of the problem and its solution encourages the creation of a common vocabulary between the end users of a system and its developers, thus creating a shared understanding of the problem being solved.*
  - *"The use of continuous integration creates opportunities to recognize risk early and make incremental corrections without destabilizing the entire development effort.*
  - *"An object-oriented architecture provides a clear separation of concerns among the disparate elements of a system, creating firewalls that prevent a change in one part of the system from rending the fabric of the entire architecture." [Booch]*

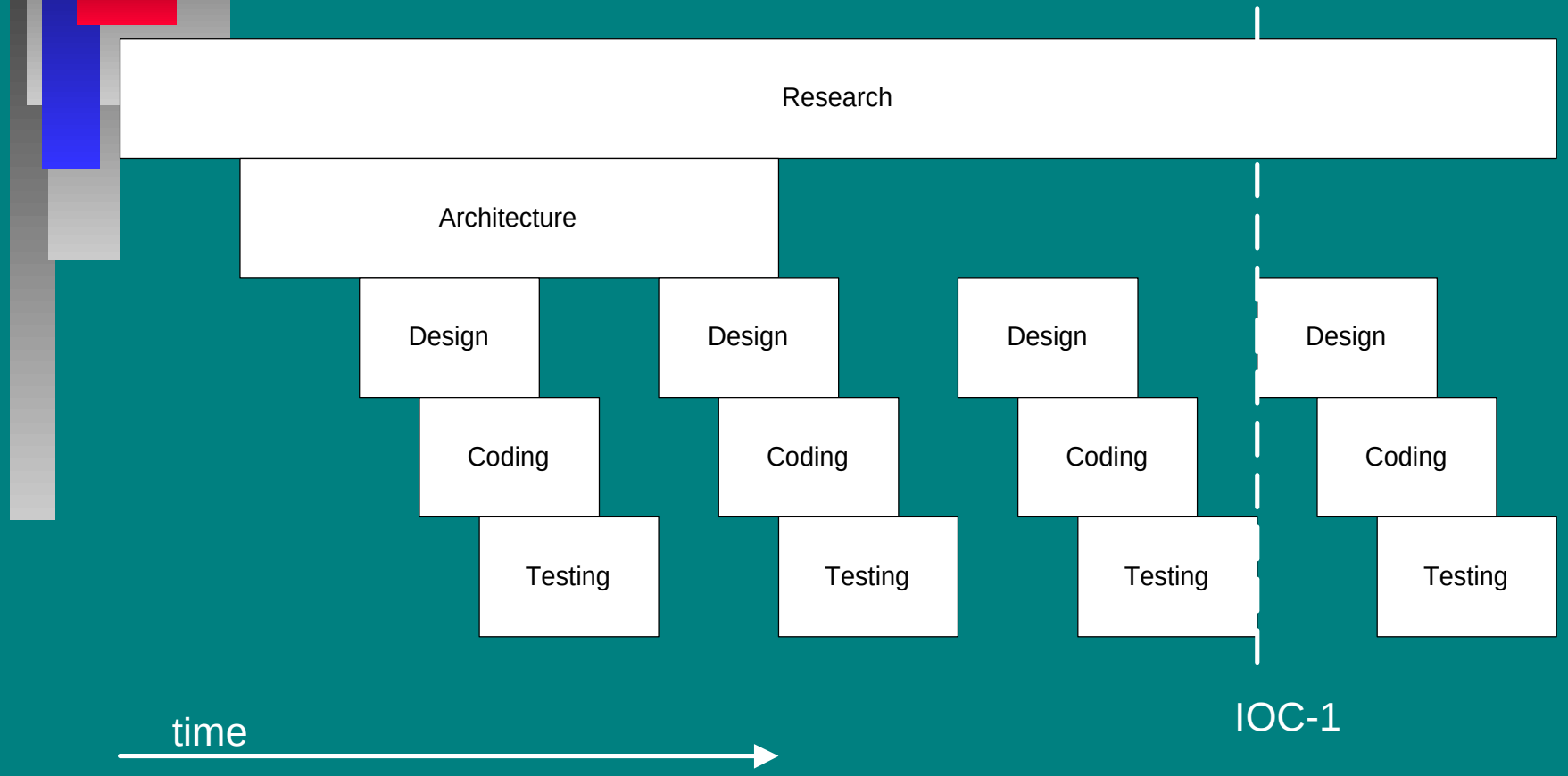
Development Process

- *analysis*
  - *requirements definition*
  - *domain analysis*
  - *use cases / scenarios*
- *design*
  - *architectural design*
  - *class design*
- *coding*
- *quality assurance*
  - *tests*
  - *inspections and reviews*
  - *metrics*

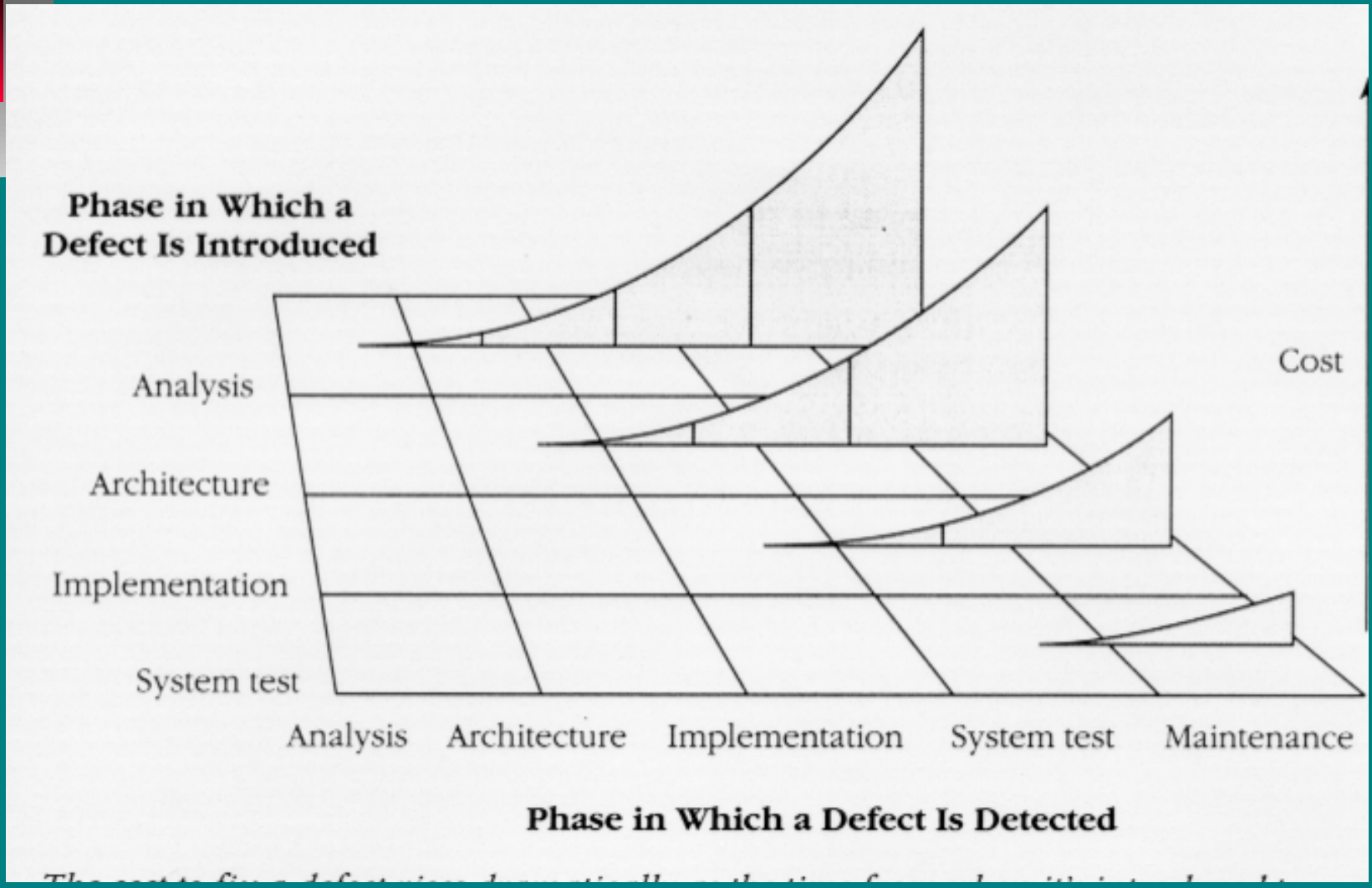
Iterative Process Flow Chart [Kahn]



TRANSIMS Development Process



Cost of Defect [McConnell]



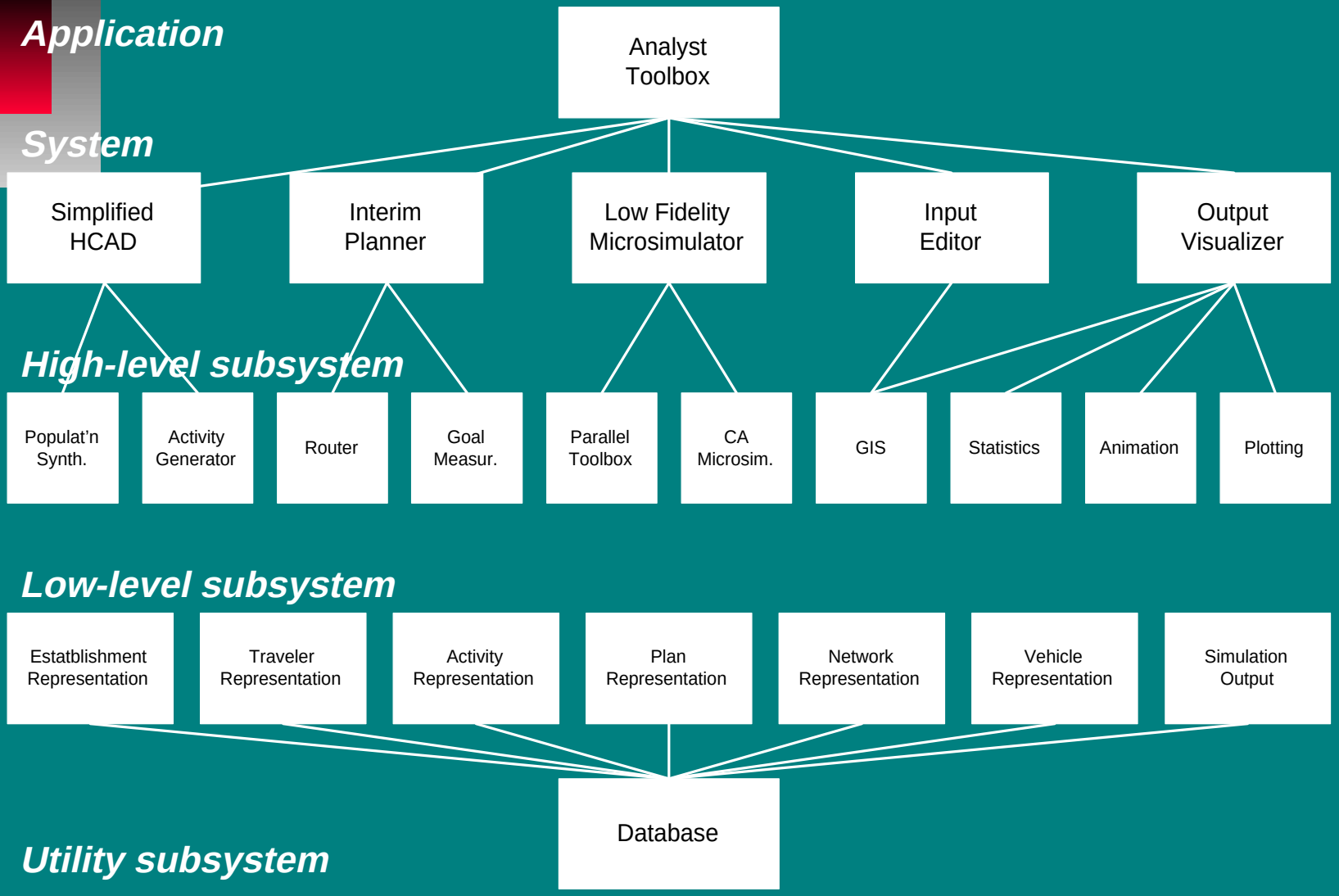
Analysis

- *"*Object-oriented analysis *is a method of analysis that examines requirements from the perspective of classes and objects found in the vocabulary of the problem domain." [Booch]*
- *"***Domain analysis** *attempts to understand the basic abstractions in a discipline. The goal of domain analysis is to determine a general domain model from which it is possible to develop multiple applications. . . . The outcome of a domain analysis is the identification of reuse opportunities across applications in a domain." [Goldberg]*
- *Use cases are a tool for the* **definition of requirements** *and the* **analysis of the problem domain**. *Requirements are generated both from the customer's definition of the problem and results of analyzing the abstractions in the problem's domain.*

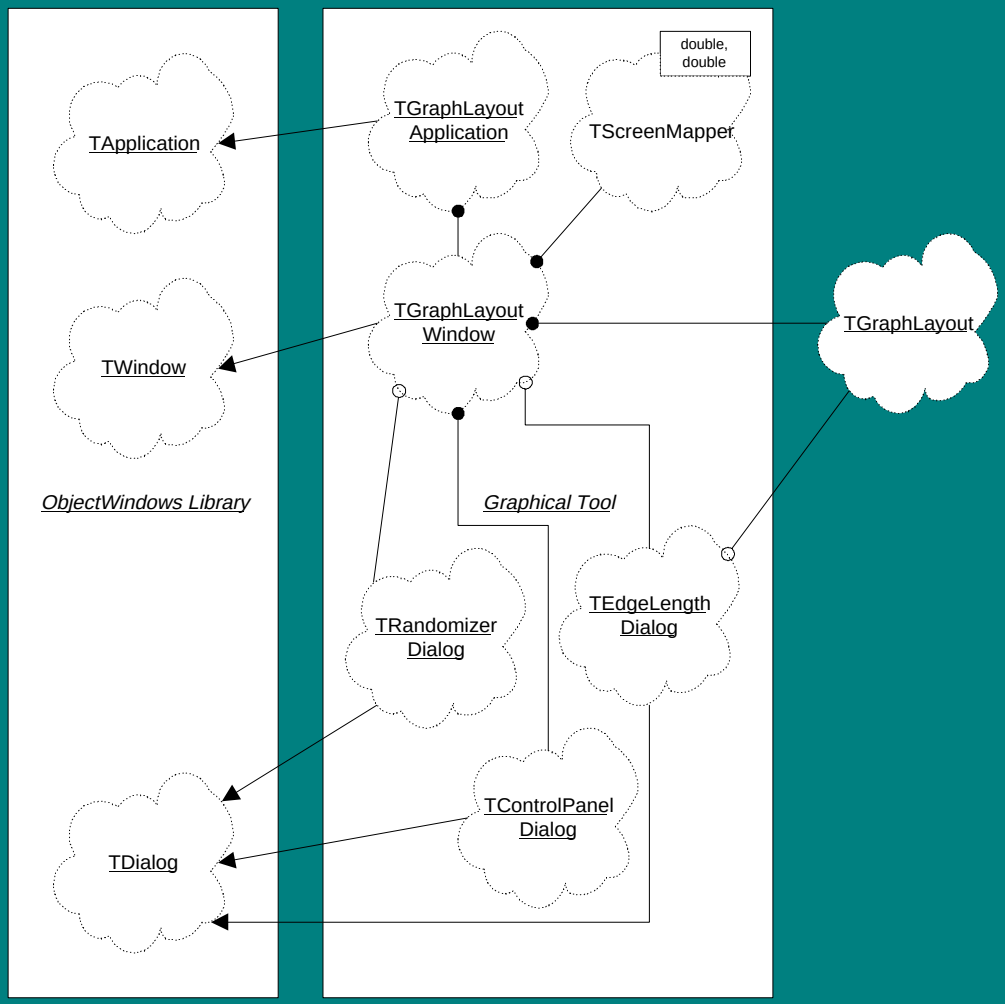
Design

- *design process*
  - *“Identify the classes and objects at a given level of abstraction.*
  - *“Identify the semantics of these classes and objects.*
  - *“Identify the relationship among these classes and objects.*
  - *“Specify the interface and then the implementation of these classes and objects.” [Booch]*
- *architectural design principles*
  - *layering*
  - *modularity*
  - *use of frameworks*
- *class design principles*
  - *coupling/cohesion*
  - *completeness*
  - *patterns*
- *diagramming*

TRANSIMS Architecture



Example Class Diagram (from Graph Layout Tool)



Programming Languages

- *C++*
- *Smalltalk*
- *CLOS*
- *Java*
- *Eiffel*
- *Objective C*
- *Simula*

Programming Language Comparison [Goldberg]

Concept/ Mechanism	Benefits	Drawbacks	Languages
Object Abstraction			
Classes or templates	Capture similarity among like objects	Overhead for applications that have many one-of-a-kind objects	C++, CLOS, Eiffel, Objective C, Smalltalk
Encapsulation			
Multiple levels	Flexibility in controlling visibility	Reduces potential for reuse	C++, CLOS
Circumvention	Potential performance boost by avoiding message-passing as a way of accessing data	Violates an object's encapsulation and introduces tight coupling between objects	C++, CLOS, Objective C
Polymorphism			
Unbounded polymorphism	Flexibility in prototyping and maintenance to replace an object with another object that supports the required interface	Inhibits static type checking	CLOS, Objective C, Smalltalk
Bounded polymorphism	Provides additional information for type checking and optimization	Reduces the flexibility of object references	C++, Eiffel
Inheritance			
Of interface specification without implementation	Promotes behavior reuse and object substitution	In isolation no drawback, but if there is no implementation inheritance, then forces redundant coding	C++, Eiffel
Of implementation	Promotes code reuse	Inheritance hierarchies may not reflect object type specializations	C++, CLOS, Eiffel, Objective C, Smalltalk
Multiple	Useful when a class is viewed as a combination of two or more different superclasses	Can lead to exceedingly complex inheritance patterns, difficult to understand and maintain	C++, CLOS, Eiffel[1]

[1] A number of add-on packages provide multiple inheritance for Smalltalk programmers, although these are not widely used.

continued

Concept/ Mechanism	Benefits	Drawbacks	Languages
Typing			
No declarations	Less work for the developer	Omits important information that could improve implementation understandability	CLOS, Smalltalk
Formal declarations	Makes implementations easier to understand and provides necessary information for static type checking	More work for the developer	C++, Eiffel
Static type checking	Detects type errors before execution	May impede prototyping by rejecting implementations that could run	C++, Eiffel, Objective C
Dynamic type checking	Allows flexible construction and testing of implementations	Detects type errors only at runtime	CLOS, Objective C, Smalltalk
Binding			
Static	Avoids runtime lookup, or use of large amounts of memory to store compiled code for alternative execution pathways	Requires unique names for all system operations, and may require multiple code changes when requirements change	(C and Pascal)
Dynamic	Creates very flexible code that is resilient to the addition and removal of types	Incurs the overhead of binding at execution time, or the creation of extra code for alternative execution pathways	CLOS, Smalltalk
Both	Can choose the appropriate form of binding for the situation	Requires the developer to know the difference and to specify the information needed to support both	C++, Eiffel, Objective C
Object Lifetime			
Classes are objects available at runtime	Additional abstraction capability and runtime flexibility to modify and add classes	Overhead for maintaining the class information in the runtime environment	CLOS, Objective C, Smalltalk
Manual runtime storage reclamation	Allows the developer to control reclamation in special situations	Is error prone and forces the developer to deal with a low-level systems issue	C++, Objective C
Automatic runtime storage reclamation	Frees the developer from determining when space is to be reclaimed	Imposes an overhead on the runtime system to do the reclamation	CLOS, Eiffel, Smalltalk

Bibliography: Management

- *G. Booch*, Object Solutions: Managing the Object-Oriented Project, *(Menlo Park, California: Addison-Wesley, 1996).*
- *A. Goldberg and K. S. Rubin*, Succeeding with Objects: Decision Frameworks for Project Management, *(Reading, Massachusetts: Addison-Wesley, 1995).*
- *I. Jacobson, M. Christerson, P. Johsson, and G. Övergaard*, Object-Oriented Software Engineering: A Use Case Driven Approach, *(Wokingham, England: Addison-Wesley, 1992).*

Bibliography: Design

- *G. Booch*, Object-Oriented Analysis and Design with Applications, *(Redwood City, California: Benjamin/Cummings, 1994).*
- *D. Collins*, Designing Object-Oriented User Interfaces, *(Redwood City, California: Benjamin/Cummings, 1995).*
- *E. Gamma, R. Helm, R. Johnson, and J. Vlissides*, Design Patterns: Elements of Reusable Object-Oriented Software, *(Reading, Massachusetts: Addison-Wesley, 1995).*
- *M. Lorenz*, Object-Oriented Software Development, *(Englewood Cliffs, New Jersey: Prentice Hall, 1993).*
- *J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen*, Object-Oriented Modeling and Design, *(Englewood Cliffs, New Jersey: Prentice Hall, 1991).*
- *N. Wilkinson*, Using CRC Cards, *(New York: SIGS Books, 1995).*

Bibliography: Coding

- *R. G. G. Cattel*, Object Data Management, *(Reading, Massachusetts: Addison-Wesley, 1994).*
- *S. Khoshafian, Object-Oriented Databases, (New York: John Wiley & Sons, 1993).*
- *W. LaLonde*, Discovering Smalltalk, *(Redwood City, California: Benjamin/Cummings, 1994).*
- *M. Lorenz*, Rapid Software Development with Smalltalk, *(New York: SIGS Books, 1995).*
- *R. C. Martin*, Designing Object-Oriented C++ Applications Using the Booch Method, *(Englewood Cliffs, New Jersey: Prentice-Hall, 1995).*
- *R. Otte, P. Patrick, and M. Roy*, Understanding CORBA, *(Upper Saddle River, New Jersey: Prentice-Hall, 1996).*

Bibliography: Quality Assurance

- *T. Gilb and D. Graham*, Software Inspection, *(Wolkingham, England: Addison-Wesley, 1993).*
- *S. H. Kan*, Metrics and Models in Software Quality Engineering, *(Reading, Massachusetts: Addison-Wesley, 1995).*
- *S. Maguire*, Debugging the Development Process, *(Redmond, Washington: Microsoft Press, 1994).*
- *S. McConnell*, Code Complete, *(Redmond, Washington: Microsoft Press, 1993).*